**CSCI 5922 Exam**

**Honor Statement:**

**I, Casey Cooper, understand that I will do this exam alone and without the help of other humans. I understand exactly what it means to “do my own work” and I will do my own work. If I fail to do this and/or my work looks similar to other work, I understand that very bad things can happen.**

**Signature \_Casey Cooper\_\_\_\_\_\_\_\_Date\_\_\_\_\_\_12/7/2023\_\_\_\_\_\_**

**Directions: Please complete this Exam using THIS DOCUMENT.**

**Download and save this document.** Place all answers, work, illustrations, images, etc. that you want graded into this document.

If you are asked to code, you can create you code in Jupyter, as .py, on git, or whatever. You just need to have a link to your code that you will place on this document. Therefore, when you submit this exam, you will submit only this document.

**Please save this document as: YourName\_NN\_Exam\_2023.docx.**

**Notes and Rules:**

1. **No questions are permitted** by anyone for any reason during this exam. Follow the instructions, do not overcomplicate things, and make assumptions (that you clearly write down) if/as needed. Part of the test is your ability to “do the test”.
2. **It is not permitted to work with any other humans** on this Exam. While I am not concerned about this as I feel that all of you are very ethical, I am required to note that anyone who works together must get a “0” grade and may both fail the class and potentially have further issues with the program. Just do your own work 😊
3. **This Exam is open** – meaning you can use the web, class notes, my website, my code, your code, etc. **If you are in doubt about something – do not use it**. For example, if your buddy Bob posts code on the web that answers one of these questions and then you use it – that’s cheating. Using my code or your code is fine.
4. **This Exam will be due no later than 12/10 (Sunday) by 11:59 pm MT**. If you wait until the last hour to submit and run into a problem, there will be no solution. This Exam cannot be late for any reason. Please do not test this. Submit EARLY – like 24 hours early! I have set up the submission area so that you can **submit as many times as you want.** So, you can submit in advance (like Saturday), then you can do more work on Sunday if you wish, and then you can submit again if you want. **We will use the LAST (latest) submission** for grading. This way, there is no need to wait until the last minute to submit. Again, to be clear, the **system will close and will lock at 11:59pm MT on 12/10. Exams not submitted by that time will not be graded.** Also – for those of you who always push things – if you submit the “wrong version” or a “poem for your honey” instead of the Exam – that will **not matter!** **BE CAREFUL**. **Submit early and submit correctly.** (It makes me sad that I have to say all of this 😐)

**Part 1: Common Interview Questions** and Learning New Things. This part of the Exam will ask you to define new terms and/or answer questions. Please keep your answers brief, succinct, concise, and precise, and never more than **5 sentences**. (more than 5 sentences can result in points lost)

Think of this as an interview.

1. What is an activation function (in neural networks) and why is it used?

**An activation function in neural networks is a function it takes a linear input and produces a nonlinear output. Without an activation function, a neural network would just be linear regression. So, activation functions are used because they allow neural networks to produce a nonlinear prediction for classification based on a linear input. There are many types of activation functions, but some of the most common ones are the sigmoid, ReLU, or tanh.**

1. What is an exploding gradient and give an example of what could cause it.

**An exploding gradient is when very large updates are made to the weights of a neural network after back propagation. This can cause a neural network to become unstable and perform poorly. Exploding gradients are caused by multiplying gradients with values greater than one through neural network layers during back propagation. Exploding gradients can be solved by reducing the number of layers in the neural network, choosing different activation functions, choosing a different type of neural network, or doing weight regularization.**

1. When using an activation function in a CNN that predicts images, why might you choose the ReLU?

**ReLU Is commonly used in CNNs because it speeds up training and prevents vanishing gradients. The ReLU activation function makes computation scale linearly with the size of the CNN. Using sigmoid or tanh in a CNN can result in vanishing gradients. ReLU doesn’t have this issue.**

1. A transformer (such as for language translation) has an encoder and a decoder. Suppose you have a word that is one-hot encoded. What 4 things will a common encoder include when embedding that word? Hint: The first one is some kind of embedding. What are the other three?

**A common encoder will include a positional embedding, and a weight for the key, query, and value of that word. K, Q, and V matrices are calculated for every word in the input sequence at the same time. Then each word is encoded with a positional embedding and the respective K, Q, and V value from each matrix.**

1. Define BERT (Bidirectional Encoder Representations from Transformers).

**BERT is an encoder only transformer that is trained bi-directionally. This means the entire input sequence is read at once which allows the model to learn the context of a word based on all of the surrounding words. Since BERT Is an encoder only model, its goal is not to generate text. Instead it produces representations of texts that can be used for many different NLP tasks such as sentiment analysis, NLI, or NER.**

1. What is cross attention in a transformer?

**Cross attention is when the encoder passes the key and value vectors to the decoder. The decoder then uses the key and value vectors along with its own query vector to then generate an output. Since encoder decoder models generate one word at a time the decoder’s input is the sequence that the model has generated, while the encoders input is the entire sequence. Cross attention allows the decoder to gain information about the entire input sequence, which is important to determining the probability of what the next word that is generated should be.**

1. What is transfer learning?

**Transfer learning is the use of a pre trained model on a different problem than it was originally trained for. For example, you could take a model that was pre trained on classifying images of cars and use it as a starting point for classifying images of trucks. Transfer learning is very useful as you don't have to develop the whole architecture of the model. It's also helpful, since some models require an enormous amount of computing power to be trained effectively. Instead of having to do this every time, the model can just be pre trained once and then fine-tuned on the specific problem.**

1. Define GAN (Generative adversarial network)?

**A GAN involves two neural networks where one is the generator and one is the discriminator. The generator is a convolutional neural network and the discriminator is a deconvolutional neural network. The goal of the generator is to produce artificial data that is hard to distinguish from real data and the goal of the discriminator is to distinguish the real data from the artificially generated data. Typically a GAN is used for image generation.**

1. Define GPT.

**GPT stands for generative pre trained transformer. It is a large language model that makes use of the transformer architecture to generate text. The text is generated by predicting the probability of what the next word should be based on we provided input.**

1. Define ChatGPT.

**ChatGPT is a large language model developed by OpenAI that makes use of transformers. It can generate text based on the prompt provided by the user. It can be used for a wide range of applications such as question answering or classification.**

1. Common activation functions include ReLU, sigmoid, tanh, and softmax (among others). Give an example when you would use the sigmoid as the last activation function in a NN. Give an example when you would use softmax as the last activation function in a NN.

**You would use the sigmoid activation function as the last activation function in a neural network when there are only two classes to predict. Sigmoid pushes values towards zero or one making it useful to differentiate between two classes. You would use the softmax as the last activation function in a neural network when there are more than two classes to predict. The softmax can produce probabilities over any number of classes, making it useful to make predictions when there are more than two classes.**

1. Suppose you have labeled input data where the labels are one-hot encoded. Suppose also that your labels can be one of three categories (like dog, cat, mouse for example). Next, suppose the last activation function of your NN is the softmax. Which Loss function would you choose to use in this case and why?

**You would use categorical cross entropy is the loss function in this case. The first reason is that there are three categories to predict and categorical cross entropy can account for three or more classes. Categorical cross entropy is also specifically used when labels are one hot encoded. If the labels were just integer values, then sparse categorical cross entropy would be used.**

1. Why use max pooling CNNs – what does max pooling do?

**Max pooling is used to reduce complexity of the CNN model while also Keeping only the most important features at the same time. The way max pooling works is that the image matrix is broken up into equally sized sections. Then the maximum value is kept from each section and all the others are discarded. The result is a new smaller matrix, that only contains the maximum or most important values from each section of the original image matrix.**

**Part 2: Architectures - Derivatives - and Keras**

For this part of the exam, you will illustrate your understanding of the basic/common architectures and in simple cases the derivatives for back propagation. This part of the exam will ask you to “draw” architectures. You may do this using the “draw” option in Word, by using “shapes” in Word, or if you must, by hand (and then insert your drawing). You may NOT copy/paste from the web. Why? Because I want YOU to create. Here you will also be asked to “fully label” your networks. What this means is that you will note in a smart way where all the weights are, where the biases are, what the activation functions are, and what the loss function is. You will also properly note/label hidden layers and units, or for the CNNs the filters and kernels, etc. In other words, all the parts of the networks should be designated. Finally, you will be asked to show the derivative(s). When asked this, read what you are being asked to do and then meet that requirement. **The best thing to do is to show all your work and as much as you can.** Remember, not showing something can mean one of two things. It can mean that its just too obvious to show or it can mean that you do not know it and so did not show it. Unfortunately, the graders must assume the latter. So, be safe, and show your work.

**Question 1**

1. Create and fully label an ANN that has an input for 4D data and has two hidden layers. The first hidden layer will have 4 units, the second hidden layer will have three units, and the output will have three units. For the first hidden layer, use the Sigmoid. For the second hidden layer use the ReLU, and for the output layer use the softmax. Use the categorical cross entropy as your Loss function.

A diagram of a complex network

Description automatically generated with medium confidence

1. Choose any weight between your first and second hidden layers and circle it – so we know the one you selected. Next, write out the derivatives in the chain that you would need to calculate the update for that weight. Once you write out the chain of derivatives, then derive the gradient equation. When you do this – show your work. What are the values of each derivative? Note – make any assumption you feel you need. (Do not overcomplicate this. You are just writing out the chain rule of derivatives and then showing their values.) **Notes: You are not using any numbers here, so your “answers” will not be numbers.**

**I chose W211.**

**dL/dW211 = dZ211/dW211 \* dH211/dZ211 \* dZ311/dH211 \* dL/dZ311**

**= H111 \* ReLU\_Derivative(Z211) \* W311 \* y^11 – y11**

**The gradient equation for just W211 would be: (H111.T @ ReLU\_Derivative(Z211)) \* ((y^11-y11) @ W311)**

1. Create and **paste here** the Keras code that would create this exact network. I prefer that you use the Keras API sequential model which starts with:

**My\_NN\_Model = tf.keras.models.Sequential([ ….**

However, there are many ways to show/do this in Keras and you may choose how to show your model.

Include the model definition, the model summary, and the model compile. You do not need to fit or run the model. For “compile” you choose what you want in there.

**Definition:**

**My\_NN\_Model = tf.keras.models.Sequential([**

**tf.keras.layers.Dense(4, input\_dim=4, activation='sigmoid'),**

**tf.keras.layers.Dense(3, activation='relu'),**

**tf.keras.layers.Dense(3, activation='softmax')**

**])**

**Summary:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**dense (Dense) (None, 4) 20**

**dense\_1 (Dense) (None, 3) 15**

**dense\_2 (Dense) (None, 3) 12**

**=================================================================**

**Total params: 47 (188.00 Byte)**

**Trainable params: 47 (188.00 Byte)**

**Non-trainable params: 0 (0.00 Byte)**

**Compile:**

**My\_NN\_Model.compile(**

**loss="categorical\_crossentropy",**

**metrics=["accuracy"],**

**optimizer='adam'**

**)**

**Question 2**

Create a Neural Network that takes in a single greyscale image that is 30 by 30. The network will have two filters, each that use a different 3 by 3 kernel to filter the image using the Keras Conv2D default for the stride and the padding as “same”. The next step in the network will perform 2 by 2 max pooling. The next step will use 4 filters (each with its own 3 by 3 kernel – stride (1,1) – and padding = “same”). The next step will be 2 by 2 max pooling. The next step will flatten and send into a fully connected NN. The output will have three units (three outputs are generated) and will use softmax. All other activation functions (except the last which is softmax) will be ReLU.

1. Draw this architecture. Make sure it is fully labeled with all sizes and shapes as well as other critical elements.

A diagram of a mathematical equation

Description automatically generated with medium confidence

1. Create this architecture in Keras and paste the code here. Specifically, you will need to paste the code your used to define the model, to summarize the model, and to compile the model. In the compile portion, you can use “adam”, “categorical\_crossentropy”, and “accuracy” for your metrics. You will not include fitting or running the model. Please also paste the results of your model summary here.

**Definition:**

**CNN\_Model = tf.keras.models.Sequential([**

**tf.keras.layers.Conv2D(input\_shape=(30, 30, 1), kernel\_size=(3,3), filters=2, padding='same', activation="relu"),**

**tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),**

**tf.keras.layers.Conv2D(filters=4, kernel\_size=(3, 3), padding='same', strides=(1, 1), activation='relu'),**

**tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),**

**tf.keras.layers.Flatten(),**

**tf.keras.layers.Dense(3, activation='relu'),**

**tf.keras.layers.Dense(3, 'softmax')**

**])**

**Summary:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Layer (type) Output Shape Param #**

**=================================================================**

**conv2d\_2 (Conv2D) (None, 30, 30, 2) 20**

**max\_pooling2d\_2 (MaxPoolin (None, 15, 15, 2) 0**

**g2D)**

**conv2d\_3 (Conv2D) (None, 15, 15, 4) 76**

**max\_pooling2d\_3 (MaxPoolin (None, 7, 7, 4) 0**

**g2D)**

**flatten\_1 (Flatten) (None, 196) 0**

**dense\_5 (Dense) (None, 3) 591**

**dense\_6 (Dense) (None, 3) 12**

**=================================================================**

**Total params: 699 (2.73 KB)**

**Trainable params: 699 (2.73 KB)**

**Non-trainable params: 0 (0.00 Byte)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Compile:**

**CNN\_Model.compile(optimizer='adam',**

**loss="categorical\_crossentropy",**

**metrics=['accuracy'])**

**Part 3: Applying Neural Nets (ANN, CNN, LSTM) to real labeled text data.**

For this part of the Exam, I have gathered articles on three topics: football, science, and politics. The data has already been cleaned, tokenized, and vectorized. Each row (vector) in the dataset is an article, each row is also labeled as football, science, or politics. Each column is a word in the vocabulary. The data itself represents the number of times each word appear in that given article. (The data was gathered from newsapi.org) .

**Here is a link to the cleaned, prepared, labeled data.**

<https://drive.google.com/file/d/1-ZAbxWN29iCo44kaLSfYmV2E8YDKcgGE/view?usp=sharing>

---

(If you want to know how this was done (**not required**) – here is code and a tutorial) <https://gatesboltonanalytics.com/?page_id=254>

---

**The overall goals here include:**

1. Coding, comparing, and using an ANN, CNN, and LSTM RNN in TF/Keras (Python) to Train models and to Test their accuracy.
2. You want to see if you can predict the topic of an article (in this case – football, science, or politics).
3. You also want to compare and illustrate the accuracy of your models and determine/discuss which model (ANN, CNN, or LSTM) is best and why this might be.
4. **It is up to you how to do this and how best to illustrate and explain your steps, results, and conclusions. Assume the reader is non-technical.**
5. You will include a **link** to your code, but do **not paste or otherwise include code on this Exam** document. (Again, you can place your code wherever you want as long as there is a link to it)

**Specific Requirements:**

There are many ways to do this. The following offers a few core requirements. Beyond this, **YOU must decide what to do and how best to do it**. Part of your grade will be based on your flow, discussion, illustrations, report, and communication of methods and results. Again, you will post a link to the code, but you will not include or paste code here.

1. Use Python and TF/Keras to Train and then Test the accuracy for an ANN, CNN, and LSTM RNN. In other words, you will use three different Neural Networks to create models that should predict whether a test vector (which represents an article on a topic) is on the topic of *science*, *football*, or *politics*. You will need to write code to do this. You already have code for ANNs, CNNs, and LSTM RNNs, so you may choose to repurpose/update your code as needed.
2. To show your work and to **illustrate and explain** your work, results, and conclusions you must include at least the following:
3. A link to your code. If you wish, you can put your code on your website, Google Colab, GitHub or wherever, and then include the URL here.
4. Show and explain how you **prepared the data** so that you can use it properly to Train and Test your models. (You are not required to validate – but you certainly can). Specifically, if you split the data, discuss and illustrate this. If you encode the labels, discuss and illustrate this, etc. Use images (like screenshots) as needed. YOU decide and explain/show what you are doing.
5. **DO NOT** include or paste any code. You do not need nor should you use “code” to explain or illustrate what you are doing. Use illustrations, images, explanations. Pretend that the person grading this paper does not know Python but does want to see and understand what you did, what you found, how your models compare, which model worked best, etc.
6. TO be clear - You will be coding, training, and then testing three types of models: ANN, CNN, LSTM. Therefore, you should include screen images (small portions) of the training for each (a few of the last epochs), as well as **confusion matrices** for each that illustrate the test data accuracy for each model.
7. Discuss and describe what you are doing and showing.
8. Discuss and illustrate the results. Which model worked best (have confusion matrices that support this discussion). Comment on which model you expected to work the best, which model actually worked the best and why.

**From here, you determine what is needed.**

You are welcome to use my code to assist (or other examples from the web or the Keras website)

**DATA PREPARATION**

The data that was provided to us was already cleaned. So, there were only two things that needed to be done in order to prepare it for neural network modeling. First, the labels had to be converted to numeric values. Initially, the labels were science, football, and politics. However, the labels need to be converted to numeric values for a neural network. So, all of the football labels were converted to 0, all of the politics labels we're converted to a 1, and all of the science labels were converted to a 2. The second thing that needed to be done was splitting the initial data into train, validation, and test sets. The splits ended up being approximately 80% of the data for the train set, 10% of the data for the validation set and 10% of the data for the test set. After the data was split, each set had to have the labels separated from the rest of the data.

**Initial Data:** The initial data contained 1493 news articles that were either classified as football, politics, or science. The 300 most common words from all of the articles were kept and are represented by the columns in the dataframe apart from the label column. The following is an image of the initial dataframe.

A screen shot of a computer

Description automatically generated

**Train Set:** The train set contains 1193 rows from the original data frame. This makes up approximately 80% of the data. The train set is split up into two data frames where one only has the label column and the other has every other column representing the words. The following are images of the labels and text for the training set.

**Train Text:** Has 1193 rows and 300 columns, where each row represents a news article, and each column represents one of the 300 most common words found in all the articles.

A screenshot of a computer

Description automatically generated

**Train Labels:** Has 1193 rows and 1 column, where each row contains a corresponding label for the training text.

A screenshot of a computer

Description automatically generated

**Test Set:** The test set contains 150 rows from the original data frame. This makes up approximately 10% of the data. The test set is split up into two data frames where one only has the label column and the other has every other column representing the words. The following are images of the labels and text for the test set.

**Test Text:** Has 150 rows and 300 columns, where each row represents a news article, and each column represents one of the 300 most common words found in all the articles.

A screenshot of a computer

Description automatically generated

**Test Labels:** Has 150 rows and 1 column, where each row contains a corresponding label for the test text.

A screenshot of a computer

Description automatically generated

**Validation Set:** The validation set contains 150 rows from the original data frame. This makes up approximately 10% of the data. The validation set is split up into two data frames where one only has the label column and the other has every other column representing the words. The following are images of the labels and text for the validation set.

**Validation Text:** Has 150 rows and 300 columns, where each row represents a news article, and each column represents one of the 300 most common words found in all the articles.

A screenshot of a computer

Description automatically generated

**Validation Labels:** Has 150 rows and 1 column, where each row contains a corresponding label for the validation text.

A black rectangular object with white lines

Description automatically generated

**ANN MODELING**

**Initial Model**

**Training Epochs:**

Epoch 1/100

38/38 [==============================] - 1s 17ms/step - loss: 1.0979 - accuracy: 0.3529 - val\_loss: 1.0818 - val\_accuracy: 0.3200

Epoch 2/100

38/38 [==============================] - 0s 5ms/step - loss: 1.0590 - accuracy: 0.5080 - val\_loss: 1.0506 - val\_accuracy: 0.5067

Epoch 99/100

38/38 [==============================] - 0s 5ms/step - loss: 0.3229 - accuracy: 0.8843 - val\_loss: 0.7689 - val\_accuracy: 0.6467

Epoch 100/100

38/38 [==============================] - 0s 4ms/step - loss: 0.3218 - accuracy: 0.8852 - val\_loss: 0.7719 - val\_accuracy: 0.6467

**Loss Plot:**

**A graph of a line

Description automatically generated with medium confidence**

**Accuracy Plot:**

**A graph of a line

Description automatically generated with medium confidence**

**Test Accuracy: 0.7933**

**Test Loss: 0.5353**

Looking at the loss plot for the initial model, the loss for the validation set starts increasing around the 40th epoch. Even before that, the validation loss stops decreasing at the same rate as the training loss around the 10th epoch. Looking at the accuracy plot for the initial model, validation accuracy starts to decrease around the 40th epoch. Even before that, the validation accuracy stops increasing as much as the training accuracy. Both of these plots are indicative of overfitting. Preventing overfitting, as well as increasing the validation accuracy will be the goals of the final model. Looking at the test accuracy, it's almost 10% higher than the validation accuracy. The test accuracy would be expected to be similar to the validation accuracy. One of the reasons this might not be the case and this instance, is that there isn't a ton of data. This causes the validation and test sets to be somewhat small which could cause accuracy differences between them.

**Final Model**

**Training Epochs:**

Epoch 1/100

38/38 [==============================] - 1s 12ms/step - loss: 1.4038 - accuracy: 0.3277 - val\_loss: 1.2409 - val\_accuracy: 0.3600

Epoch 2/100

38/38 [==============================] - 0s 6ms/step - loss: 1.2306 - accuracy: 0.3277 - val\_loss: 1.1428 - val\_accuracy: 0.3600

Epoch 45/100

38/38 [==============================] - 0s 7ms/step - loss: 0.5452 - accuracy: 0.7795 - val\_loss: 0.6735 - val\_accuracy: 0.6867

Epoch 46/100

38/38 [==============================] - 0s 6ms/step - loss: 0.5538 - accuracy: 0.7720 - val\_loss: 0.6738 - val\_accuracy: 0.6800

**Loss Plot:**

**A graph of a line graph

Description automatically generated with medium confidence**

**Accuracy Plot:**

**A graph of a line

Description automatically generated with medium confidence**

**Test Accuracy: 0.8133**

**Test Loss: 0.5310**

**Confusion Matrix:**

**A screenshot of a computer screen

Description automatically generated**

The final model looks much better overfitting wise and slightly better accuracy wise. Looking at the accuracy and loss plots for the final model, the validation accuracy and validation loss follow the training much better. An early stopping method was used to stop training more epochs before overfitting occurred, which is why the final model is trained on less epochs. Looking at the final model, the validation accuracy at the final epoch is slightly higher than the validation accuracy at the final epoch of the initial model. The validation accuracy increased from 0.6467 to 0.6800. The test accuracy for the final model is also a little bit higher than the test accuracy for the initial model. The test accuracy increased from 0.7933 to 0.8133. So, both goals of reducing overfitting from the initial model and improving accuracy from the initial model were achieved.

**CNN MODELING**

**Initial Model**

**Training Epochs:**

Epoch 1/100

38/38 [==============================] - 2s 28ms/step - loss: 1.0289 - accuracy: 0.4619 - val\_loss: 0.8900 - val\_accuracy: 0.6333

Epoch 2/100

38/38 [==============================] - 1s 16ms/step - loss: 0.7461 - accuracy: 0.7024 - val\_loss: 0.7440 - val\_accuracy: 0.7000

Epoch 99/100

38/38 [==============================] - 1s 19ms/step - loss: 0.2402 - accuracy: 0.8977 - val\_loss: 1.8349 - val\_accuracy: 0.6267

Epoch 100/100

38/38 [==============================] - 1s 19ms/step - loss: 0.2344 - accuracy: 0.9078 - val\_loss: 1.8749 - val\_accuracy: 0.6200

**Loss Plot:**

**A graph of a number of people

Description automatically generated with medium confidence**

**Accuracy Plot:**

**A graph of a graph

Description automatically generated with medium confidence**

**Test Accuracy: 0.7400**

**Test Loss: 1.1454**

Looking at the loss plot for the initial model, the loss for the validation set starts increasing almost immediately after the first several epochs. Looking at the accuracy plot for the initial model, validation accuracy starts decreasing almost immediately after the first several epochs. Both of these plots are indicative of overfitting. Since the model is overfitting so quickly the complexity of the model likely needs to be reduced. Preventing overfitting, as well as increasing the validation accuracy will be the goals of the final model. Looking at the test accuracy, it's approximately 10% higher than the validation accuracy. The test accuracy would be expected to be similar to the validation accuracy. One of the reasons this might not be the case and this instance, is that there isn't a ton of data. This causes the validation and test sets to be somewhat small which could cause accuracy differences between them.

**Final Model**

**Training Epochs:**

Epoch 1/100

38/38 [==============================] - 2s 22ms/step - loss: 1.0990 - accuracy: 0.3252 - val\_loss: 1.0982 - val\_accuracy: 0.3067

Epoch 2/100

38/38 [==============================] - 0s 8ms/step - loss: 1.0974 - accuracy: 0.3378 - val\_loss: 1.0947 - val\_accuracy: 0.3067

Epoch 17/100

38/38 [==============================] - 0s 13ms/step - loss: 0.5482 - accuracy: 0.7661 - val\_loss: 0.6835 - val\_accuracy: 0.6933

Epoch 18/100

38/38 [==============================] - 0s 8ms/step - loss: 0.5472 - accuracy: 0.7678 - val\_loss: 0.7054 - val\_accuracy: 0.6867

**Loss Plot:**

**A graph of a graph

Description automatically generated with medium confidence**

**Accuracy Plot:**

**A graph of a person and person

Description automatically generated**

**Test Accuracy: 0.7800**

**Test Loss: 0.5766**

**Confusion Matrix:**

**A screenshot of a computer screen

Description automatically generated**

The final model looks much better overfitting wise and slightly better accuracy wise. Looking at the accuracy and loss plots for the final model, the validation accuracy and validation loss follow the training much better. An early stopping method was used to stop training more epochs before overfitting occurred, which is why the final model is trained on less epochs. Looking at the final model, the validation accuracy at the final epoch is slightly higher than the validation accuracy at the final epoch of the initial model. The validation accuracy increased from 0.6200 to 0.6867. The test accuracy for the final model is also a little bit higher than the test accuracy for the initial model. The test accuracy increased from 0.7400 to 0.7800. So, both goals of reducing overfitting from the initial model and improving accuracy from the initial model were achieved.

**LSTM MODELING**

**Initial Model**

**Training Epochs:**

Epoch 1/100

38/38 [==============================] - 16s 300ms/step - loss: 1.1011 - accuracy: 0.3227 - val\_loss: 1.0976 - val\_accuracy: 0.3333

Epoch 2/100

38/38 [==============================] - 9s 248ms/step - loss: 1.0983 - accuracy: 0.3386 - val\_loss: 1.0990 - val\_accuracy: 0.3133

Epoch 99/100

38/38 [==============================] - 8s 214ms/step - loss: 0.9672 - accuracy: 0.4962 - val\_loss: 0.9792 - val\_accuracy: 0.5133

Epoch 100/100

38/38 [==============================] - 8s 216ms/step - loss: 0.9544 - accuracy: 0.5071 - val\_loss: 0.9600 - val\_accuracy: 0.5000

**Loss Plot:**

**A graph with blue and orange lines

Description automatically generated**

**Accuracy Plot:**

**A graph of a person and person

Description automatically generated with medium confidence**

**Test Accuracy: 0.5400**

**Test Loss: 0.9490**

The validation accuracy and loss follow the training accuracy and loss, which is good because that means the model is not overfitting. However, the model performs overall very poorly. The accuracy never really goes above 0.55. There is a weird point around the 50th epoch in both the loss and accuracy plots where the model stops improving and essentially resets itself. Solving this issue and increasing the overall accuracy will be the goals of the final model. Looking at the test accuracy, it's similar to the validation accuracy, which is expected.

**Final Model**

**Training Epochs:**

Epoch 1/40

38/38 [==============================] - 21s 440ms/step - loss: 1.0994 - accuracy: 0.3252 - val\_loss: 1.0991 - val\_accuracy: 0.3333

Epoch 2/40

38/38 [==============================] - 15s 403ms/step - loss: 1.0999 - accuracy: 0.3319 - val\_loss: 1.0980 - val\_accuracy: 0.3933

Epoch 39/40

38/38 [==============================] - 10s 260ms/step - loss: 0.8443 - accuracy: 0.5859 - val\_loss: 0.9936 - val\_accuracy: 0.5067

Epoch 40/40

38/38 [==============================] - 10s 252ms/step - loss: 0.8577 - accuracy: 0.5708 - val\_loss: 1.0294 - val\_accuracy: 0.4800

**Loss Plot:**

**A graph of a line

Description automatically generated with medium confidence**

**Accuracy Plot:**

**A graph with blue and orange lines

Description automatically generated**

**Test Accuracy: 0.5200**

**Test Loss: 0.9908**

**Confusion Matrix:**

The final model has no overfitting issue and no issue with a massive decrease in performance like the initial model. However, there was very little improvement on accuracy. I had to stop training around 40 epochs to prevent overfitting. Even training for more epochs does not result in an improved validation accuracy though. The LSTM model performed very poorly. Maybe I just wasn’t able to find the correct architecture though.

**CONCLUSION**

**ANN Accuracy:** Validation: 0.6800 Test: 0.7933

**CNN Accuracy:** Validation: 0.6867 Test: 0.7800

**LSTM Accuracy:** Validation: 0.4800 Test: 0.5200

I originally thought that the ANN or LSTM models would perform the best. This is mainly because LSTM models generally perform well with text data and CNN models generally perform best with image data. However, my predictions were definitely not correct in this case. Looking at the accuracies above, the ANN model performed the best, the CNN model performed the 2nd best, and the LSTM model was the worst. The ANN and CNN models were pretty comparable in their accuracies. The LSTM model was much worse, having an accuracy that was approximately 20% worse. I’m not exactly sure why the LSTM model was so much worse. A couple possibilities could be that I just wasn’t able to find a good architecture for the model, or that there wasn’t enough data to create an effective LSTM model.